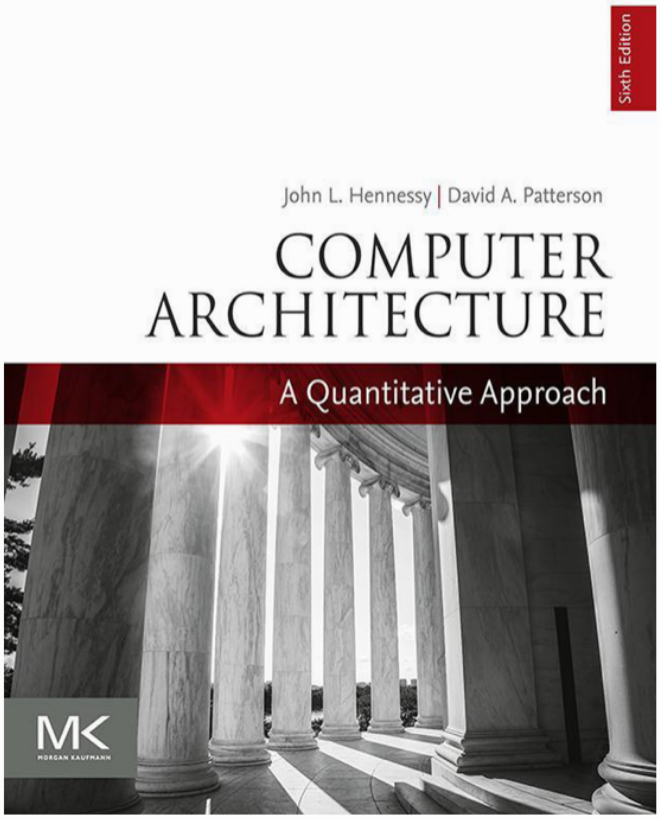
EXHIBIT D



Computer Architecture

A Quantitative Approach

SIXTH EDITION

John L. Hennessy

Stanford University

David A. Patterson

University of California, Berkeley

WITH CONTRIBUTIONS BY

Krste Asanović
University of California, Berkeley
Jason D. Bakos
University of South Carolina
Robert P. Colwell
R&E Colwell & Assoc. Inc.
Abhishek Bhattacharjee
Rutgers University
Thomas M. Conte
Georgia Tech
José Duato
Proemisa
Diana Franklin
University of Chicago
David Goldberg
eBay
Norman P. Jouppi
Google
Sheng Li
Intel Labs
Naveen Muralimanohar
HP Labs
Gregory D. Peterson
University of Tennessee
Timothy M. Pinkston

University of Southern California
Parthasarathy Ranganathan
Google
David A. Wood
University of Wisconsin–Madison
Cliff Young
Google
Amr Zaky
University of Santa Clara

Table of Contents

Although efficient, ISPs have two major downsides. Given the increasing demand for improved image quality in handheld devices, the first is the inflexibility of an ISP, especially as it takes years to design and manufacture a new ISP within an SOC. The second is that these computing resources can be used only for the image-enhancing function, no matter what is needed at the time on the PMD. Current generation ISPs handle workloads at up to 2 Tera-operations per second on a PMD power budget, so a DSA replacement has to achieve similar performance and efficiency.

Pixel Visual Core Software

Pixel Visual Core generalized the typical hardwired pipeline organization of kernels of an ISP into a *directed acyclic graph* (*DAG*) of kernels. Pixel Visual Core image-processing programs are typically written in Halide, which is a domain-specific functional programming language for image processing. Figure 7.31 is a Halide example that blurs an image. Halide has a functional section to express the function being programmed and a separate schedule section to specify how to optimize that function to the underlying hardware.

```
Func buildBlur(Func input) {
      // Functional portion (independent of target processor)
    Func blur_x("blur_x"), blur_y("blur_y");
    blur_x(x,y) = (input(x-1,y) + input(x,y)*2 + input(x+1,y)) / 4;
    blur_y(x,y) = (blur_x(x,y-1) + blur_x(x,y)*2 + blur_x(x,y+1)) / 4;

    if (has_ipu) {
      // Schedule portion (directs how to optimize for target processor)
      blur_x.ipu(x,y);
      blur_y.ipu(x,y);
}
    return blur_y;
}
```

FIGURE 7.31 Portion of a Halide example to blur an image.
The `ipu(x,y)` suffix schedules the function to Pixel Visual Core. A blur has the effect of looking at the image through a translucent screen, which reduces noise and detail. A Gaussian function is often used to blur the image.

Pixel Visual Core Architecture Philosophy

The power budget of PMDs is 6–8 W for bursts of 10–20 seconds, dropping down to tens of milliwatts when the screen is off. Given the challenging energy goals of a PMD chip, the Pixel Visual Core architecture was strongly shaped by the relative energy costs for the primitive operations mentioned in Chapter 1 and made explicit in Figure 7.32. Strikingly, a single 8-bit DRAM access takes as much energy as 12,500 8-bit additions or 7–100 8-bit SRAM accesses, depending on the organization of the SRAM. The 22 × to 150 × higher cost of IEEE 754 floating-point operations over 8-bit integer operations, plus the die size and energy benefits of storing narrower data, strongly favor using narrow integers whenever algorithms can accommodate them.

Operation	*Energy (pJ)*	Operation	Energy (pJ)	Operation	Energy (pJ)
8b DRAM LPDDR3	125.00	8b SRAM	1.2–17.1	16b SRAM	2.4–34.2
32b Fl. Pt. muladd	2.70	8b int muladd	0.12	16b int muladd	0.43
32b Fl. Pt. add	1.50	8b int add	0.01	16b int add	0.02

FIGURE 7.32 Relative energy costs per operation in picoJoules assuming TSMC 28-nm HPM process, which was the process Pixel Visual Core used [17] [18][19][20].
The absolute energy cost are less than in Figure 7.2 because of using 28 nm instead of 90 nm, but the relative energy costs are similarly high.

In addition to the guidelines from Section 7.2, these observations led to other themes that guided the Pixel Visual Core design:

- *Two-dimensional is better than one-dimensional*: Two-dimensional organizations can be beneficial for processing images as it minimizes communication distance and because the two- and three-dimensional nature of image data can utilize such organizations.
- *Closer is better than farther*: Moving data is expensive. Moreover, the relative cost of moving data to an ALU operation is increasing. And of course DRAM time and energy costs far exceed any local data storage or movement.

A primary goal in going from an ISP to an IPU is to get more reuse of the hardware via programmability. Here are the three main features of the Pixel Visual Core:

1. Following the theme that two-dimensional is better than one-dimensional, Pixel Visual Core uses a two-dimensional SIMD architecture instead of one-dimensional SIMD architecture. Thus it has a two-dimensional array of independent *processing elements* (*PEs*), each of which contains 2 16-bit ALUs, 1 16-bit MAC unit, 10 16-bit registers, and 10 1-bit predicate registers. The 16-bit arithmetic follows the guideline of providing only the precision needed by the domain.
2. Pixel Visual Core needs temporary storage at each PE. Following the guideline from Section 7.2 of avoiding caches, this PE memory is a compiler-managed scratchpad memory. The logical size of each PE memory is 128 entries of 16 bits, or just 256 bytes. Because it would be inefficient to implement a separate small SRAM in each PE, Pixel Visual Core instead groups the PE memory of 8 PEs together in a single wide SRAM block. Because the PEs operate in SIMD fashion, Pixel Visual Core can bind all the individual reads and writes together to form a "squarer" SRAM, which is more efficient than narrow and deep or wide and shallow SRAMs. Figure 7.33 shows four PEs.
3. To be able to perform simultaneous stencil computations in all PEs, Pixel Visual Core needs to collect inputs from nearest neighbors. This communication pattern requires a "NSEW" (North, South, East, West) shift network: it can shift data en masse between the PEs in any compass direction. So that it doesn't lose pixels along the edges as it shifts images, Pixel Visual Core connects the endpoints of the network together to form a torus.

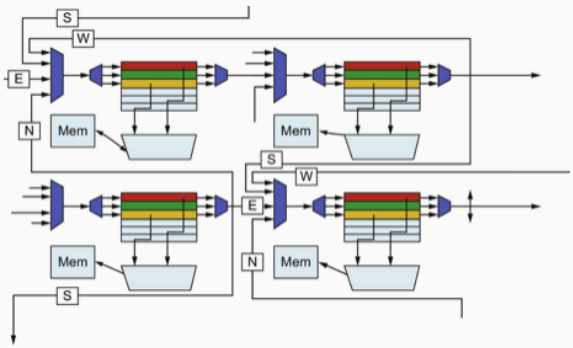


FIGURE 7.33 The two-dimensional SIMD includes two-dimensional shifting "N," "S," "E," "W," show the direction of the shift (North, South, East, West).
Each PE has a software-controlled scratchpad memory.

Note that the shift network is in contrast with the systolic array of processing element arrays in the TPU and Catapult. In this case, software explicitly moves the data in the desired direction across the